Evaluation of On-the-fly State Space Reductions*

Radek Pelánek

xpelanek@fi.muni.cz

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic

Abstract

We present an overview of equivalences of finite state structures and discuss methods for computing reduced structures on-the-fly. We evaluate merits of these reductions on a large set of model checking case studies. It turns out that the achieved reduction can be significant, but that it is not so "drastic" as sometimes claimed in the literature.

Keywords: model checking, state space reductions, experimental evaluation

1 Introduction

Explicit finite state model checking is a successful technique for verification of concurrent systems and particularly communication protocols. It is based on an exhaustive exploration of the whole state space of a given model. Even sophisticated verification techniques like automatic abstraction refinement or compositional verification are, at the end, based on the exhaustive exploration of the state space. The main limitation of the exhaustive exploration is the state explosion problem.

Specifications which are checked over the model are usually expressed in some temporal logic. Temporal logics cannot distinguish between structures which are 'equivalent'; the exact meaning of 'equivalent' depends on the logic we are interested in. Instead of checking the specification over the (very big) state space we can check it over some (smaller) equivalent structure. One possibility is to generate the whole state space, reduce it by suitable equivalence, and finally check the specification over the reduced structure. This approach, however, does not reduce the peak memory requirements which are the main practical limitation of model checking. Another possibility is to employ static analysis and use specific information about the model to compute a reduced structure on-the-fly.

There are many techniques for such on-the-fly reduction. The most well-known are symmetry reduction and partial order reduction. However, it is not clear what are the practical merits of these reductions. Researchers usually demonstrate the effectiveness of proposed reduction on just few (well selected) examples. Often we can find claims about exponential or at least "drastic" reduction. Are these claims appropriate?

In this work, we present a systematic overview of equivalences, their connections to temporal logics, and methods for the on-the-fly computation of reduced structures. We also give a realistic evaluation of merits of discussed reductions. For the evaluation we use three model checking tools and many case studies previously studied in the literature. The results show that the effect of reductions can be significant but it is not so drastic as often claimed in the literature.

In the full version of the paper [20] we give more technical description of discussed reductions and we also present some new reduction techniques.

2 Models, equivalences, and logics

Let AP be a finite set of atomic propositions. *Kripke structure* over AP is a tuple $M = (S, s_0, \to, L)$, where S is a finite set of states, $s_0 \in S$ is an initial states, $\to \subseteq S \times S$ is a transition relation, and $L : S \to 2^{AP}$ is a labeling function. As a modeling language (a symbolic representation of Kripke structure) we consider networks of extended finite state machines (EFSM); for the definition see [20].

A *path* π in a Kripke structure is a (possibly infinite) sequence $\pi = s_0 s_1 s_2 \dots$ such that $\forall i \geq 0 : s_i \to s_{i+1}$. Two paths $\pi = s_0 s_1 \dots$ and $\pi' = s'_0 s'_1 \dots$ are *equivalent* iff $\forall i \geq 0 : L(s_i) = L(s'_i)$. Paths π, π' are *stutter*

*Partially supported by the Grant Agency of Czech Republic grant No. 201/03/0509 and by the Academy of Sciences of Czech Republic grant No. 1ET408050503.

equivalent iff there exists a partition $B_1, B_2, \dots$ of π and a partition $B'_1, B'_2, \dots$ of π' such that $\forall j \geq 0$: blocks B_j, B'_j are nonempty and finite and $\forall s \in B_j, s' \in B'_j : L(s) = L(s')$. A state s is *reachable* in M iff there exists path from s_0 to s. State s is *deadlocked* iff $\forall s' \in S : s \not\to s'$. An atomic proposition $a \in AP$ (respectively deadlock) is reachable in M iff there exists a state s reachable in M such that $s \models a$ (respectively s is deadlocked).

For the rest of this section let $M = (S, s_0, \to, L)$ and $M' = (S', s'_0, \to', L')$ be two Kripke structures with the same set of atomic propositions AP. We have several notions of an equivalence between Kripke structures:

- The structures M, M' are *reachability equivalent* iff for each $a \in AP$: a is reachable in $M \Leftrightarrow a$ is reachable in M'.
- The structures M, M' are *deadlock equivalent* iff deadlock is reachable in $M \Leftrightarrow$ deadlock is reachable in M'.
- The structures M, M' are *trace equivalent* iff for each path π in M there exists an equivalent path π' in M' and vice versa.
- The structures M, M' are *stutter trace equivalent* iff for each path π in M there exists a stutter equivalent path π' in M' and vice versa.
- A relation $R \subseteq S \times S'$ is a simulation relation iff for all $(s, s') \in R$ the following hold:
  - $L(s) = L'(s')$
  - For every $s' \to s'_1$ there exists $s \to s_1$ such that $R(s_1, s'_1)$.

  A state *s simulates* s' (denoted $s \succeq s'$) iff there exists simulation R such that $(s, s') \in R$. The structure M *simulates* M' ($M \succeq M'$) iff $s_0 \succeq s'_0$. The structures M, M' are *simulation equivalent* iff $s_0 \succeq s'_0$ and $s'_0 \succeq s_0$.
- A relation $R \subseteq S \times S'$ is a bisimulation relation iff R is symmetrical and R is a simulation relation. States s, s' are *bisimilar* (denoted $s \sim s'$) iff there exists a bisimulation relation R such that $(s, s') \in R$. The structures M, M' are *bisimilar* iff $s_0 \sim s'_0$.
- A relation $R \subseteq S \times S'$ is a stutter simulation relation iff for all $(s, s') \in R$ the following hold:
  - $L(s) = L'(s')$
  - For every path π' in M' that starts in s' there is a path π in M that starts in s, a partition $B_1 B_2 \dots$ of π, and a partition $B'_1 B'_2 \dots$ of π' such that $\forall j \in \mathbb{N}$: blocks B_j and B'_j are nonempty and finite, and $\forall t \in B_j, t' \in B'_j : (t, t') \in R$.

  The structures M, M' are *stutter simulation equivalent* iff there exists stutter simulation relations R_1, R_2 such that $(s_0, s'_0) \in R_1$, $(s'_0, s_0) \in R_2$.
- *Stutter bisimulation equivalence* is defined analogically to bisimulation equivalence.

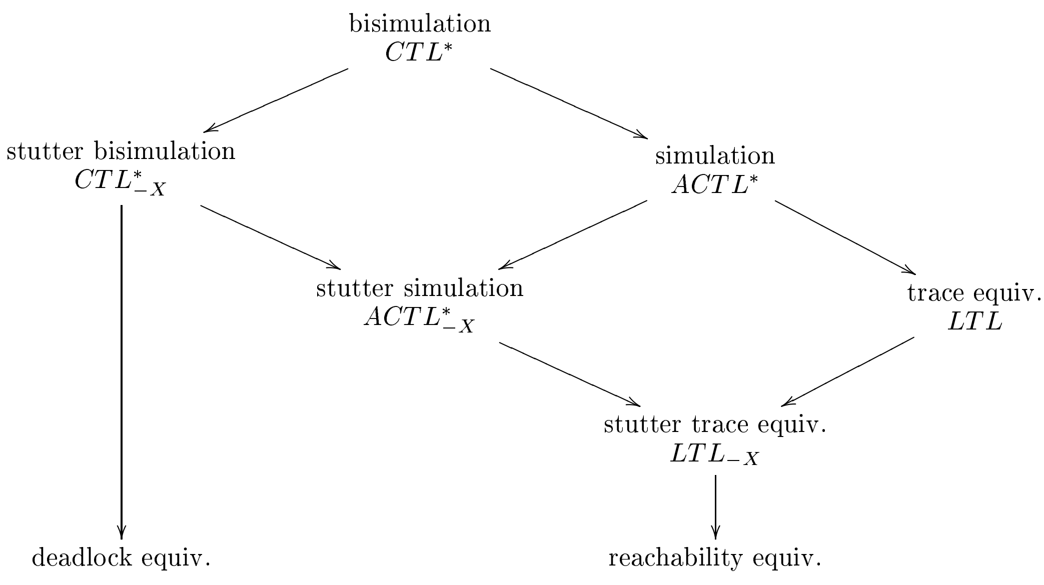

Figure 1: Relations among equivalences and temporal logics. For each equivalence we give temporal logic which is preserved by the equivalence. Proofs can be found in [4, 10, 22].

3 Computing Reductions On-the-fly

Figure 2. gives the basic algorithm GENERATE(N) which generates the reachable part of the Kripke structure. The model generated by this algorithm is bisimilar to the full structure. Our aim is to modify this basic algorithm in such a way that it will produce structure that is smaller and yet equivalent (up to one of the equivalences discussed above). In some cases we are able to perform the reduction by static transformation of the model prior to the exploration of the state space.

In this section we mention reduction preserving bisimulation and stutter equivalences. In the full version of the paper [20] we also mention some new reductions preserving reachability equivalence and deadlock equivalence.

```
1  proc GENERATE(N)
2    W = {s0}
3    while W ≠ ∅ do
4          remove s from W
5          add s to S
6          L(s) := {a ∈ AP | s ⊨ a}
7          foreach s → s' do
8            add (s, s') to R
9            if s' ∉ S ∪ W then add s' to W fi
10         od
11   od
12   return (S, s0, R, L)
13 end
```

Figure 2: The basic algorithm

3.1 Reductions Preserving Bisimulation

Reductions preserving bisimulation are based on the notion of canonization function. Let us add the line $s' := \mathit{canonize}(s')$ after the line 7 of the basic algorithm. If the *canonize* function gives unique representant from each class then we obtain the bisimulation collapse (minimal bisimulation equivalent structure). However, it is not feasible to compute a unique representant on-the-fly. For the correctness of the algorithm it is sufficient that the canonization function returns *some* bisimilar state, i.e., that $\forall s \in S : \mathit{canonize}(s) \sim s$. As long as the canonization function can be computed efficiently, the modified algorithm involves no significant overhead over the basic algorithm.

In the following we briefly discuss several ways of computing the canonization function. Note that different canonization functions can be straightforwardly combined.

Dead variables A variable is *dead* in a given state if no computation starting from this state uses the value of the variable. Dead variables can be computed (respectively safely approximated) by standard static analysis algorithms. The canonization function simply sets value of all dead variables to zero.

Equivalent values This is an extension of the idea of dead variables. Two values of a variable are equivalent in some state if the model cannot distinguish between them.

Equivalent states Similarly, we can detect equivalent states with the use of 'local bisimulation relation' on individual machines of EFSM.

Symmetry Bisimulation relation can also be identified by exploiting symmetries in the model. Symmetries are formalized by the notion of an automorphism and are usually detected with the use of permutations [15]. In the full version of the paper [20] we propose another approach for detecting automorphisms which is based on linear transformations of data values.

3.2 Reductions Preserving Stutter Equivalences

Action $\alpha \in \mathit{Act}$ is *invisible* iff $\forall s \in S : L(s) = L(\alpha(s))$. On intuitive level, invisible actions are not important for stutter equivalences. Thus we can do reductions which change the order and the execution of invisible actions. Here we discuss the main idea of several reductions based on this observation. First two reductions are 'static' (based on transformations of the model), the other three reductions are 'dynamic' (based on modification of the GENERATE algorithm).

Slicing Slicing identifies parts of the model that are relevant to the verified property [11]. Actions which cannot influence visible actions are removed by a static transformation of the model prior to the state space exploration. The resulting model is stutter trace equivalent to the original model.

Transition merging If there are two consecutive local invisible actions in the model then we can statically merge them into one atomic action. This basic idea has been formalized in different ways and under different names. Each formalization preserves some stutter equivalence: transition merging, transition compression (stutter trace equivalence) [16, 6, 12], "next" heuristics (stutter simulation) [1], path reduction (stutter bisimulation) [24].

Partial order reductions With partial order reduction we traverse only a subset of all enabled actions in a given state. We use the basic algorithm with the line 7 changed into: **foreach** $s' \in \{\alpha(s) \mid \alpha \in \mathit{ample}(s)\}$ **do** . According to the exact requirements on the function *ample* we obtain structure that can be deadlock equivalent, reachability equivalent for local properties [9], stutter trace equivalent [21], stutter simulation equivalent [22], or stutter bisimulation equivalent [8]. The basic requirement is that actions in *ample* 'commute' with other actions (this is formalized by the notion of independence).

Confluence Similar strategy is based on the identification of τ-confluent actions [2, 18] — invisible actions which satisfy some additional confluence requirements (which are similar to the independence requirements of partial order reduction). This reduction preserves stutter bisimulation.

Simultaneous reachability analysis When faced with several possible interleavings of independent and invisible actions, partial order reduction tries to traverse only one of these interleavings. Simultaneous reachability analysis rather tries to perform *all* these actions at once [17, 23], i.e., instead of executing individual actions it executes combined actions.

4 Evaluation

For most of the discussed methods it is easy to come up with an (artificial) example on which the reduction gives exponential or at least very significant improvement. Unfortunately, many authors evaluate their techniques on such examples. Moreover, reduction techniques are often evaluated on toy models with high values of parameters — this makes state spaces big and reductions significant. However, in real usage of model checkers models are complex and parameter values are low; very rarely it happens that the model is correct for low values of parameters and erroneous for high values. In [19] we argue that experiments on such toy models can lead to misleading conclusions. In this section we provide an evaluation of merits of individual reductions on realistic models. It turns out that the reduction is more temperate than often claimed.

Most of the models that we use for the evaluation are well-known model checking case studies: alternating bit protocol, Peterson's mutual exclusion protocol, bounded retransmission protocol, I-protocol, firewire link protocol, leader election protocol, real-time Ethernet protocol, cache coherence protocol, firewire tree identification protocol, file transfer protocol, X.509 authentication protocol, Needham-Schroeder protocol, production cell case study, etc. The evaluation was done with three explicit model checkers: Spin [1] (version 4.0.6), Murphi [2] (version 3.1), and DiVinE [3] (version 0.5).

We discuss to what kind of models each reduction is suitable, we summarize experimental results in other papers and report about the effect of the reduction on our models. We also discuss the run-time overhead of the reduction and the 'complexity' of its implementation. We give results only on those models on which the reduction had some effect. We present only the number of states in full and reduced structure. Reduction with respect to the number of transitions is usually very similar.

[1] `http://spinroot.com`
[2] `http://verify.stanford.edu/dill/murphi.html`
[3] `http://anna.fi.muni.cz/divine`

Dead variables (DiVinE)				POR (Spin)			
Model	Full	Reduced		Model	Full	Reduced	
rether	10,462	1,192	11.3%	cambridge	146,471	6,298	4.2%
synapse	13,973	1,981	14.1%	erathostenes	25,295	2,093	8.2%
peterson	12,498	2,376	19.0%	snoopy	61,619	9,707	15.7%
brp	4,792	1,571	32.7%	smcs	4,634	1,196	25.8%
production_cell	77,416	32,854	42.4%	mobile	30,652	9,971	32.5%
iprotocol_good	29,994	12,770	42.5%	pftp	144,813	47,356	32.7%
firewire	55,887	24,323	43.5%	i-protocol	2,207,190	919,978	41.7%
abp	11,286	5,652	50.0%	relay	876	442	50.4%
bridge	3,186	1,676	52.6%	peterson	30,432	16,720	54.9%
tip	86,556	49,082	56.7%	brp	290,174	169,208	58.3%
elevator	1,139	723	63.4%	X.509	9,028	6,094	67.5%
bakery	109,144	84,517	77.4%	sgc	299,270	293,126	97.9%
cambridge	8,592	6,962	81.0%	sliding	16,441	14,645	89.0%
resistance	151,587	129,177	85.2%	giop	638,525	638,520	99.9%

Table 1: Results for dead variables reduction and POR

4.1 Dead variables

Dead variable reduction can reduce the size of the state space up to 10% of the size of the full state space (see Table 1). Yorav [24] gives similar results on four software models. Bozga et al. [7] reports more impressive reduction but only on one parametric model. This reduction strategy is applicable to a wide class of models and it brings nearly no run-time overhead. For local variables, which are responsible for most of the reduction, we can perform the canonization by static transformation. Dead variables are easy to detect statically; it becomes more challenging only for arrays and more complex data types.

4.2 Partial order reduction

Notwithstanding the large body of theoretical work about partial order reduction methods, the number of studies concerning practical results of partial order reduction is rather small. Godefroid [9] gives evaluation on four realistic models. The sizes of reduced structures are between 3% and 55% of the size of original structure. Clarke at el. [5] reports similar results on three realistic models. Table 1. presents results of our experiments. The size of reduced structures is between 4% and 99%. Partial order reduction is applicable mainly to models with loosely coupled processes. For many of our models, which use either lot of rendezvous communication or shared variables, the method is not applicable.

The run-time overhead and complexity of static analysis depends on the quality of the reduction which we want to achieve. In our experiments we use the tool Spin. Spin uses a rather conservative approach which sacrifices some possible reduction for low overhead [13].

Confluence and simultaneous reachability analysis, other two techniques based on similar ideas as partial order reduction, have comparable results [18, 2, 17].

4.3 Symmetry reduction

Symmetry reductions based on permutations can, in theory, achieve reduction up to $n!$ where n is the number of symmetrical entities. Table 2. presents practical results. Experiments were done in the tool Murphi on the set of models which was used by Dill and Ip [15]. We have just parametrized protocols by smaller values. This gives more realistic evaluation: the size of reduced structure is between 8% and 50%. Simillar results were reported by Bosnacki et al. [3] in Symmetric Spin and by Iosif [14] for object oriented programs.

The method is, of course, applicable only to models with symmetrical entities. Typical applications are cache coherence protocols, protocols over bus with several symmetrical parties, models with several symmetrical agents. The run-time overhead is non-trivial due to the computation of canonization function. Some reduction can be sacrificed for lower overhead. The detection of symmetries can be done fully automatically only in special cases. For practical purposes it is necessary to extend the modeling language with special 'symmetric constructs' (e.g., scalarset [15]).

Permutations (Murphi)			
Model	Full	Reduced	
cache	67,418	5,629	8.3%
list4	8,893	1,489	16.7%
peterson3	882	172	19.5%
eadash	1,694	425	25.0%
sci	18,059	4,525	25.0%
ldash	740	372	50.2%

Linear transformations (DiVinE)			
Model	Full	Reduced	
cambridge	827	222	26.8%
abp	11,286	5,958	55.1%
brp	14,720	8,121	55.2%

Table 2: Results for symmetry reduction based on permutations and linear transformations

4.4 Static transformations

Our experience suggests that the effect of static transformations (merging of equivalent states, slicing, transition merging, loop acceleration) is not very dependent on the type of an application, but rather on the experience and the modeling style of a user and on possibilities of a modeling language. An experienced user can perform lot of the static transformations manually (even unconsciously). Most of our models were crafted by experienced users in rather low-level modeling formalisms and therefore these reductions are not very efficient for them. Table 3. presents results for models on which the transition merging technique was applicable. Dong and Ramakrishnan [6] report much better effect of these reduction. We suspect that this is because their modeling language does not contain atomic constructs (as opposed to Spin and DiVinE). Kurshan et al. [16], Yorav [24], and Holzmann [12] report results slightly better than we have obtained. Holzmann [12] supports our claims about the dependence on modeling style of the user as he shows that by manual re-modeling he can achieve very significant reduction.

Transition merging (Spin)			
Model	Full	Reduced	
sgc	607,750	299,270	49.2%
erathostenes	47,669	25,295	53.0%
pftp	207,481	144,813	69.7%
cambridge	166,510	146,471	87.9%
snoopy	67,656	61,619	91.0%
peterson	33,434	30,432	91.0%
smcs	5,066	4,634	91.4%
X.509	9,760	9,028	92.5%
brp	309,676	290,174	93.7%
mobile	32,668	30,652	93.8%

Transition merging (DiVinE)			
Model	Full	Reduced	
iprotocol	29,994	6,445	21.5%
rether	10,462	6,970	66.6%
resistance	151,587	108,095	71.3%
krebs	7,869	6,027	76.6%
firewire	55,887	45,155	80.8%

Table 3: Results for transition merging

We suppose that static transformations will be very important for models automatically generated from high level description languages and models created by 'naive' users who are not familiar with the underlying model checking algorithms. This type of application is becoming more and more important. In order to convincingly evaluate these techniques, it will be necessary to perform experiments on a large set of models created by non-expert users. At this moment, it is difficult to obtain such a set.

5 Summary

In this paper we provide a catalog of on-the-fly state space reduction techniques and a realistic evaluation of merits of these techniques. Our results can be summarized as follows:

- Each technique is applicable to some class of models. Nothing works really universally. More specialized techniques yield better reduction.
- On real models, no single reduction is able to reduce the size of the state space significantly under 5%. Claims about drastic reduction, which occur in some papers, are not really appropriate.
- Since there are many techniques and many of them are orthogonal, most models can be reduced quite significantly. Certainly, it is useful to implement these reductions.
- The reduction factor depends not only on application domain, but also on the user's modeling style. Some reductions, particularly those based on static transformation of the model, do not bring nearly

no improvement when applied to models crafted by expert users but can be very efficient on models created by non-expert users.

References

[1] R. Alur and B.-Y. Wang. “Next” heuristic for on-the-fly model checking. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 98–113, 1999.

[2] S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proc. of Computer Aided Verification (CAV 2002)*, number 2404 in LNCS, pages 596–609, 2002.

[3] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric spin. In *Proc. of SPIN Workshop*, volume 1885 of *LNCS*, pages 1–19, 2000.

[4] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59(1-2):115–131, 1988.

[5] E.M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(3):279–287, November 1999.

[6] Y. Dong and C. R. Ramakrishnan. An optimizing compiler for efficient model checking. In *Proc. of FORTE XII/PSTV XIX*, pages 241–256. Kluwer, B.V., 1999.

[7] J.-C. Fernandez, M. Bozga, and L. Ghirvu. State space reduction based on live variables analysis. *Sci. Comput. Program.*, 47(2-3):203–220, 2003.

[8] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time model checking. *Information and Computation*, 150(2):132–152, 1999.

[9] P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.

[10] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[11] J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. *Higher Order Symbol. Comput.*, 13(4):315–353, 2000.

[12] G. J. Holzmann. The engineering of a model checker: the gnu i-protocol case study revisited. In *Proc. of SPIN Workshop*, volume 1680 of *LNCS*, 1999.

[13] G. J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. of Formal Description Techniques VII*, pages 197–211. Chapman & Hall, Ltd., 1995.

[14] R. Iosif. Symmetric model checking for object-based programs. Technical Report TR 2001-5, Kansas State University, 2001.

[15] C. Norris Ip and David L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1–2):41–75, 1996.

[16] R. P. Kurshan, V. Levin, and Hüsnü Yenigün. Compressing transitions for model checking. In *Proc. of Computer Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, pages 569–581, 2002.

[17] K. Ozdemir and H. Ural. Protocol validation by simultaneous reachability analysis. *Computer Communications*, 20:772–788, 1997.

[18] G. J. Pace, F. Lang, and R. Mateescu. Calculating tau-confluence compositionally. In *Proc. Computer Aided Verification (CAV 2003)*, volume 2725 of *LNCS*, pages 446 – 459, 2003.

[19] R. Pelánek. Typical structural properties of state spaces. In *Proc. of SPIN Workshop*, volume 2989 of *LNCS*, pages 5–22, 2004.

[20] R. Pelánek. On-the-fly state space reductions. Technical Report FIMU-RS-2005-03, Masaryk University Brno, 2005.

[21] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Proc. of Computer Aided Verification (CAV 1994)*, volume 818 of *LNCS*, pages 377–390, 1994.

[22] W. Penczek, M. Szreter, R. Gerth, and R. Kuiper. Improving partial order reductions for universal branching time properties. *Fundamenta Informaticae*, 43(1-4):245–267, 2000.

[23] H. van der Schoot. Partial-order verification in spin can be more efficient. In *Proc. of SPIN Workshop.* Twente University, 1997.

[24] K. Yorav. *Exploiting Syntactic Structure for Automatic Verification.* PhD thesis, The Technion – Israel Institute of Technology, 2002.